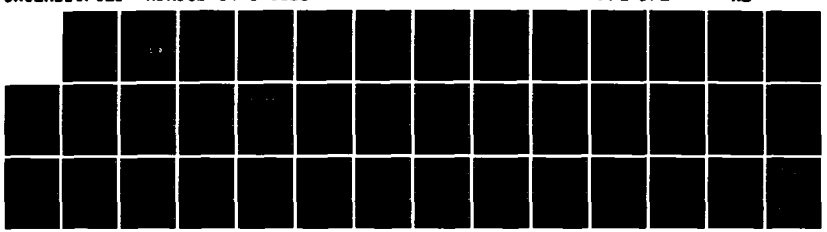
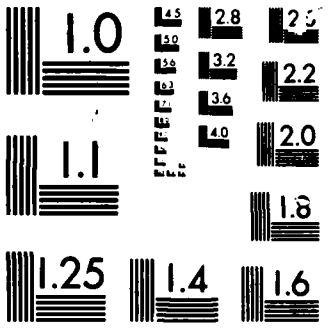
AD-A166 610 DEVELOPMENT OF A BUTTERFLY (TRADEMARK) MULTIPROCESSOR TEST BED: THE BUTTERFLY SWITCH(U) BOLT BERANEK AND NEWMAN INC CAMBRIDGE MA 30 OCT 85 BBN-5874 1/1

UNCLASSIFIED MDA903-84-C-0033 F/G 9/1 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY
CHART

BBN Laboratories Incorporated

A Subsidiary of Bolt Beranek and Newman Inc.

Report No. 5874

Quarterly Technical Report No. 3
April 16, 1984-July 15, 1984
Development of a Butterfly Multiprocessor Test Bed
The Butterfly Switch

October 1985

Prepared for:
Defense Advanced Research Projects Agency
Engineering Applications Office

AD-A166610

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188
Exp Date Jun 30, 1986

1a REPORT SECURITY CLASSIFICATION: Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT: Distribution Unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S): 5874	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION: Bolt Beranek and Newman Inc / 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION: Defense Advanced Research Projects Agency
6c ADDRESS (City, State, and ZIP Code): 10 Moulton Street, Cambridge, MA 02238	7b ADDRESS (City, State, and ZIP Code): 1400 Wilson Blvd., Arlington, VA 22209
8a NAME OF FUNDING/SPONSORING ORGANIZATION / 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: MDA903-84-C-0033
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
Development of a Butterfly Multiprocessor Test Bed: The Butterfly Switch

12 PERSONAL AUTHOR(S)

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Quarterly Technical	FROM 4/16/84 TO 7/15/84	1935, October, 30	36

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Description of the Butterfly Switch.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL: Edward A. Starr	22b TELEPHONE (Include Area Code): (617) 497-3307 / 22c OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

Quarterly Technical Report No. 3
April 16, 1984 - July 15, 1984

DEVELOPMENT OF A BUTTERFLY™ MULTIPROCESSOR TESTBED

The Butterfly Switch
October 1985

Prepared for:

Dr. Clinton Kelly, Director
Defense Advanced Research Projects Agency
Engineering Applications Office
Arlington, Virginia 22209

Table of Contents

Butterfly is a trademark of Bolt Beranek and Newman Inc.

FIGURES

TABLES

1. Overview and Summary

This technical report is the third in a series that describes progress on the development of a 128-processor Butterfly™ testbed. As part of the testbed effort, we have developed the Butterfly Switch chip: a single-chip implementation of the algorithm used to route messages through the Butterfly Switch. In conjunction with the switch chip, we have developed the packaging and a printed circuit board needed to incorporate the switch chip into Butterfly systems. This report first reviews the design of the Butterfly Switch and then describes the Butterfly switch chip and associated packaging.

It has been nearly a decade since BBN began studying the idea of using the Butterfly Switch as the basis for a parallel processor system. The design of the switch was first presented in BBN Report No. 3501, and updated in Report No. 4098. This section reviews and updates the material in those reports. The Butterfly switch is one of many multiprocessor interconnection networks that have been proposed and studied in recent years. The switch that we have built can be distinguished from many of these networks by several criteria:

- The Butterfly Switch is used purely for communications. There are no processing elements in the switch. This differentiates the Butterfly Parallel Processor from machines such as the NYU Ultracomputer where some computations are performed at the switch nodes, or the CalTech Cosmic cube, where the processors perform both computation and routing functions.

- Messages in the Butterfly Switch are independent of each other. A message is sent whenever a processor makes a memory reference or invokes a communication primitive. Furthermore, the configuration of the switch is not tuned for any particular algorithm.

- The entities that are interconnected are complete processors that operate independently. That is, we are using the Butterfly Switch to interconnect processors in a Multiple Instruction, Multiple Data Stream (MIMD) machine.

The Butterfly Switch is a data communications structure in a general sense. The services that it provides, such as addressing, flow control, and arbitration are those normally associated with a data communications network. The important parameters of the switch, such as throughput, latency, geographic extent, and protocols, are also typical of data commuunications network. The design and implementation of the Butterfly Switch borrows heavily from techniques used in packet switching networks.

The Butterfly Switch may be further characterized by some observations about its structure: first, the switch is self-routing. Each Processor Node has one switch port through which it sends all transactions, regardless of destination, and another port through which transactions arrive for that node and no other. Second, control, routing, and data transfer in the switch are inherently serial operations. This simplifies packaging and provides many opportunities for pipelining. Finally, the structure of the switch is highly regular. This facilitates VLSI implementation and further simplifies packaging.

These characteristics are not unique to the Butterfly Switch. Other types of interconnection networks, such as the crossbar, have similar properties. However, the choice of the Butterfly structure has turned out to be a good one, as it keeps the number of Switch Nodes relatively small without sacrificing performance.

The remainder of this report reviews the basic operation of the Butterfly Switch and introduces several important variations that effect its cost, performance, and flexibility. It also describes the design of the Butterfly switch chip, a VLSI implementation of the basic routing element. The scope of this report is limited to the Butterfly Switch. Discussion of other elements of the Butterfly Parallel Processor is included only when needed to illustrate the operation of the switch.

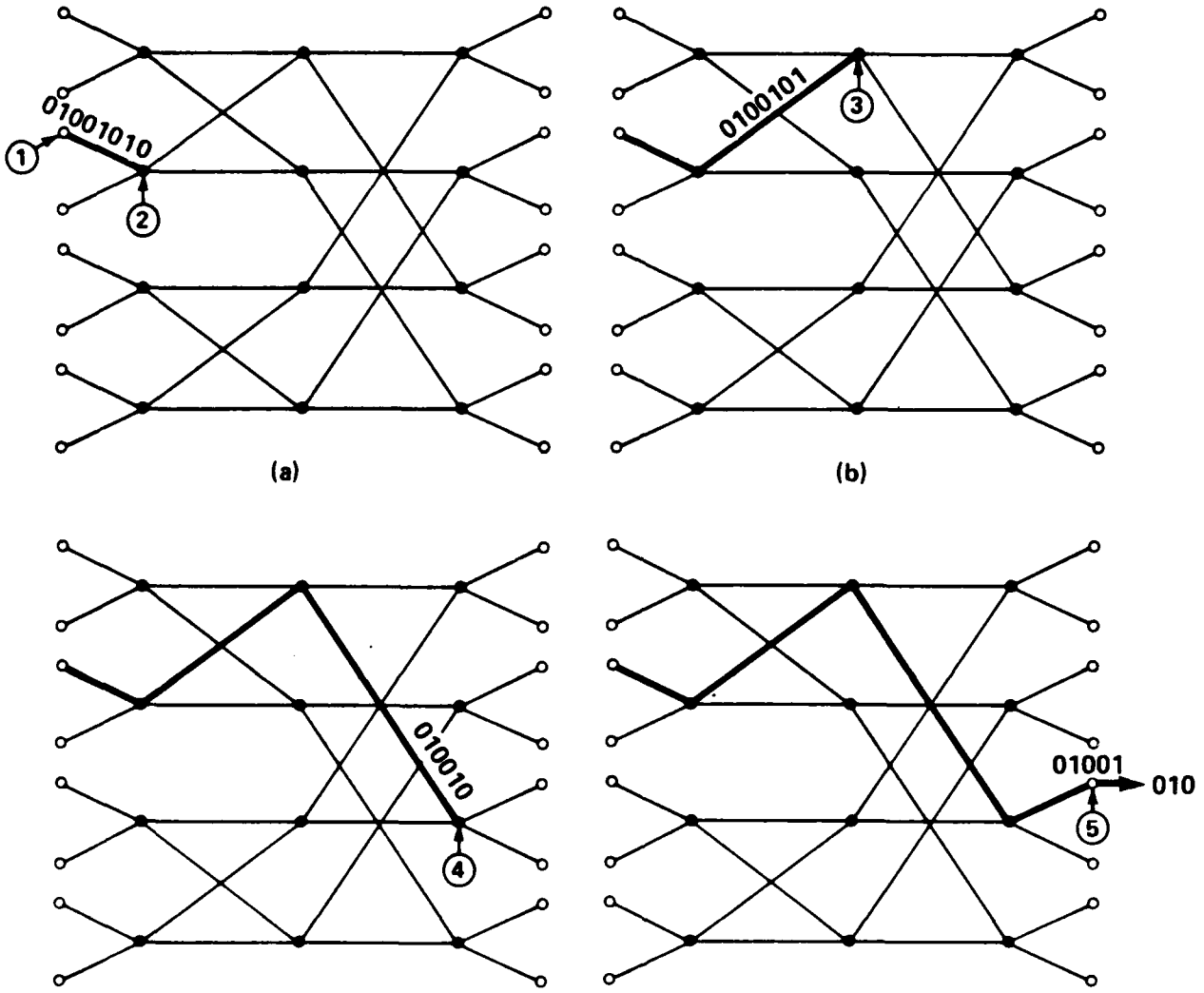
2. Switch Operation

This section presents the basic techniques used in the Butterfly Switch. Note that most of the variations described in Section 6 have been incorporated into the implementation, so that the examples shown in this section do not correspond exactly to the implementation.

The example shown in Figure 1 illustrates the operation of the Butterfly Switch. In the figure, eight sources (the circles on the left) are connected to eight destinations (the circles on the right). Each of the twelve black dots represents a switch element. The function of an element is to examine packets, which enter on either of its two left-hand inputs, and route them to the appropriate output as determined by the first bit of the packet (0=upper, 1=lower). The first bit of the packet is then discarded and the remaining bits are passed on to the next element.

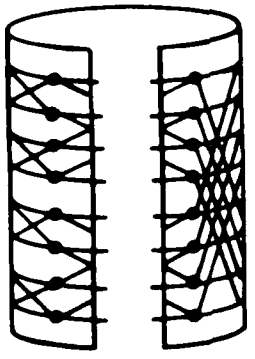
Consider the short (eight-bit) packet entering the switch at the entry port labeled 1. Since there are eight possible destinations, three bits are needed in order to select the appropriate one. These are the first three bits of the packet (the header bits), leaving five "data" bits. In (a), the packet encounters the first switch element (labeled 2) and the first header bit is removed. Since the first header bit is zero, the packet is routed to the upper output as shown in (b). At Node 3, the first bit is one; thus the lower path is chosen as shown in (c). At Node 4, the first bit is zero, so the upper path is chosen. This leads the packet to the exit port labeled 5 in (d). All that remains is to clock the five data bits through.

If the transaction had entered the switch at some other port, it would still have exited at the port labeled 5. This is easy to verify from the figure. In every case, the address of an exit port is independent of the entry port used.



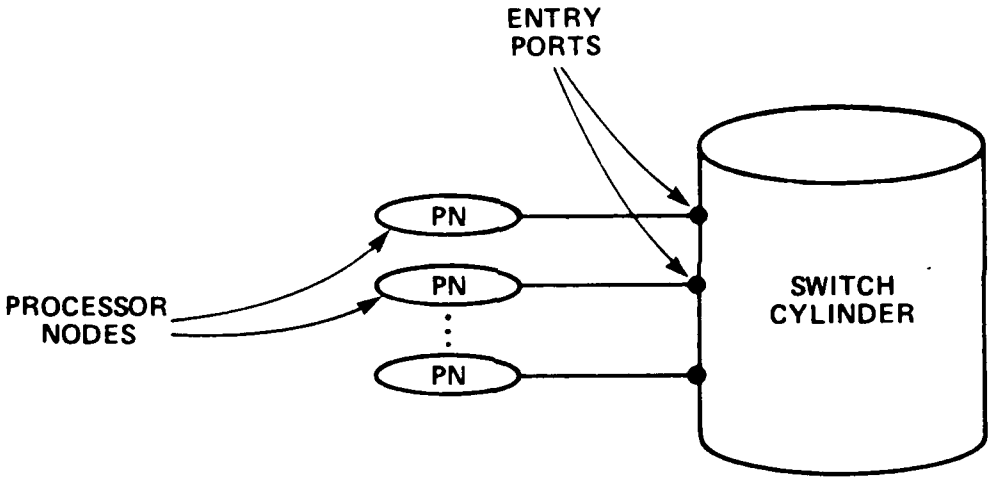
The Butterfly Switch
Figure 1

So far, we have shown a structure in which transactions flow in only one direction. To be useful, the switch must be able to carry messages in both directions, so that it can support transactions such as memory read operations. To accomplish this, one could superimpose a second switch aimed in the opposite direction to handle return traffic, or one could make the data channels through the switch bidirectional. The approach used in the current implementation of the Butterfly Switch is to bring the input ports around so that they are physically adjacent to the output ports, as shown in Figure 2. Now the switch takes on the appearance of a cylinder. The processors and memories are located along the generatrix of the cylinder; sending messages to the right and receiving messages from the left. All transactions flow in a counterclockwise direction, as viewed from the top of the cylinder.



The Butterfly Switch as a Cylinder
Figure 2

In addition to giving each processor and its memory the ability to send and receive messages, this configuration places processors and memories physically adjacent to each other. The current Butterfly Switch implementation takes the next step and combines each processor with one memory to form a unit called a "Processor Node." The Processor Nodes interface to switch ports, and the switch cylinder becomes a high bandwidth communications path between Processor Nodes. This arrangement is shown schematically in Figure 3.



Processor Nodes Connected to a Butterfly Switch Cylinder
Figure 3

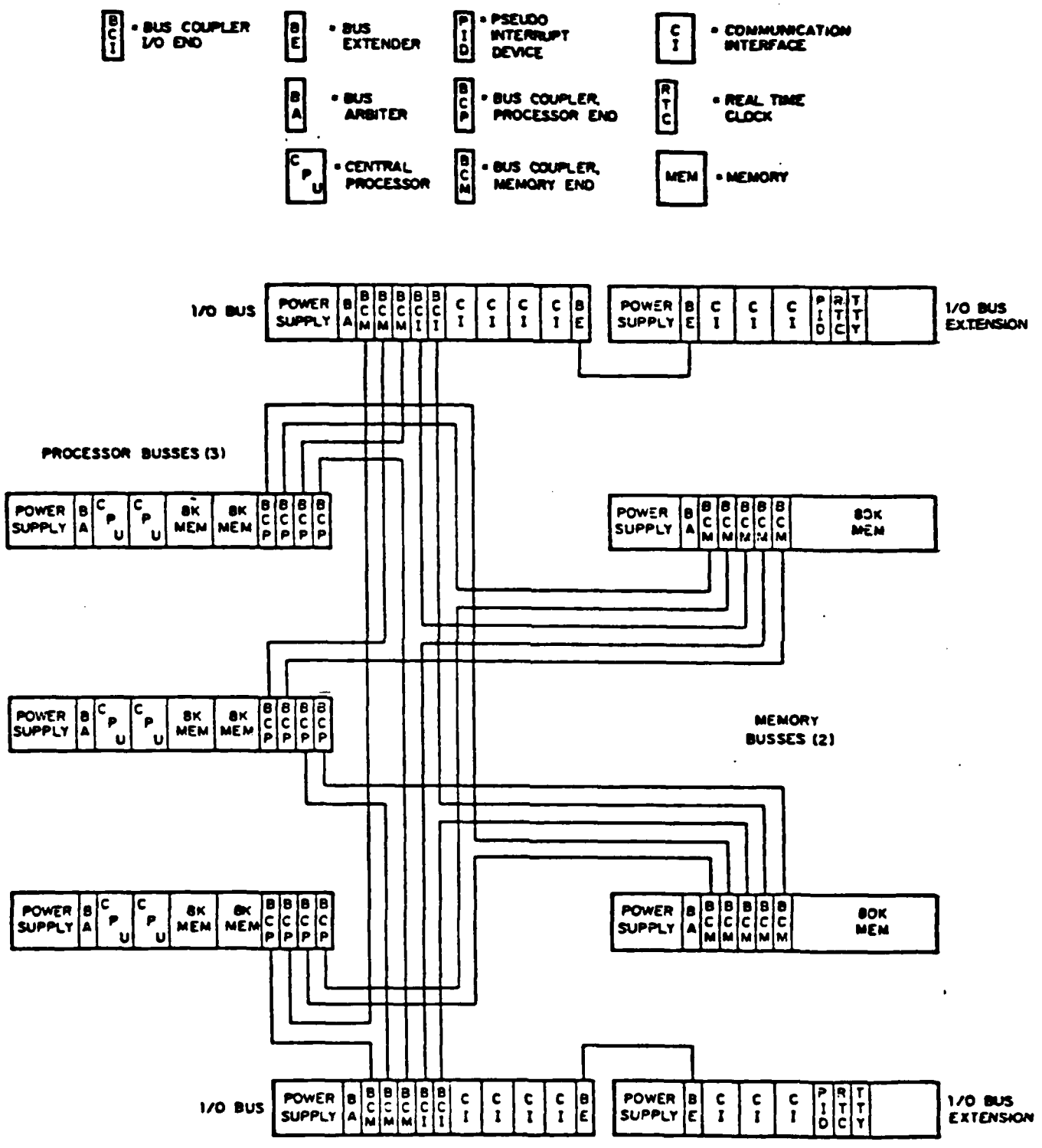
Even though processors and memories are packaged together, all of the memory in the machine is globally accessible. In addition, there is an opportunity for each processor to use a direct path to physically adjacent (i.e., local) memory, without going through the switch.

3. Comparison to Other Networks

To understand the capabilities of the Butterfly Switch, it is useful to look at the problem of connecting many processors to many memory units. Broadly speaking, interconnection networks of this kind fall on a spectrum between "completely connected" and "singly connected."

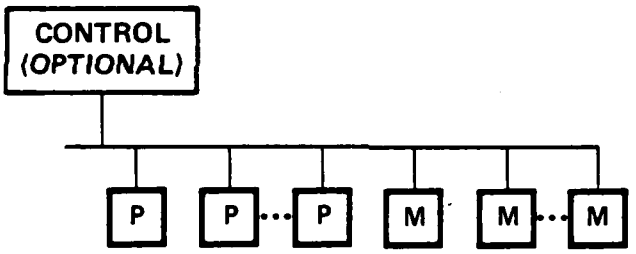
Completely connected switches are characterized by independent paths between every possible source and every possible destination. The crossbar is the most common example of this kind of switch. Another example is the Bus Coupler switch used in the Pluribus Multiprocessor [HEAR73] illustrated in Figure 4.

Potentially, the completely connected approach provides high bandwidth, since each source has its own private path to every destination. It is often said that this kind of switch is non-blocking, since there is never any contention for switch resources. In the context of multiprocessor interconnection networks, however, this characterization is somewhat misleading, since it is often the case paths will be blocked because their destinations are busy. The principal drawback of the fully connected switch is that its cost increases as the product of the number of sources and destinations. As a result, the cost of the switch can dominate overall system cost for large configurations.



Pluribus Bus Coupler Switch
Figure 4

A singly connected switch, on the other hand, has only one path connecting all of the elements, as shown in Figure 5. The path is shared by several sources according to some discipline such as time division multiplexing or asynchronous arbitration. There are many examples of this kind of switch, including computer buses [IEEE83], the Ethernet, and broadcast satellite channels [ABRA69]. The cost of a bus goes up linearly with the number of elements that it connects, so it does not have the cost explosion problem of the crossbar. However, the amount of traffic that a single bus can carry does not increase as more elements are added. The capacity of the bus therefore sets a hard limit on the number of elements that can be supported efficiently.



Singly Connected Switch
Figure 5

The Butterfly Switch represents a useful middle ground between these two extremes. For an N-port Butterfly Switch, the number of wires and Switch Nodes grows as NlogN, which contrasts favorably with the N^2 growth seen in a fully connected switch. Furthermore, the capacity of the Butterfly Switch grows linearly with the number of ports. When a new port is added to the switch, a new row of Switch Nodes is also added. This contrasts favorably with the fixed capacity of a singly connected switch.

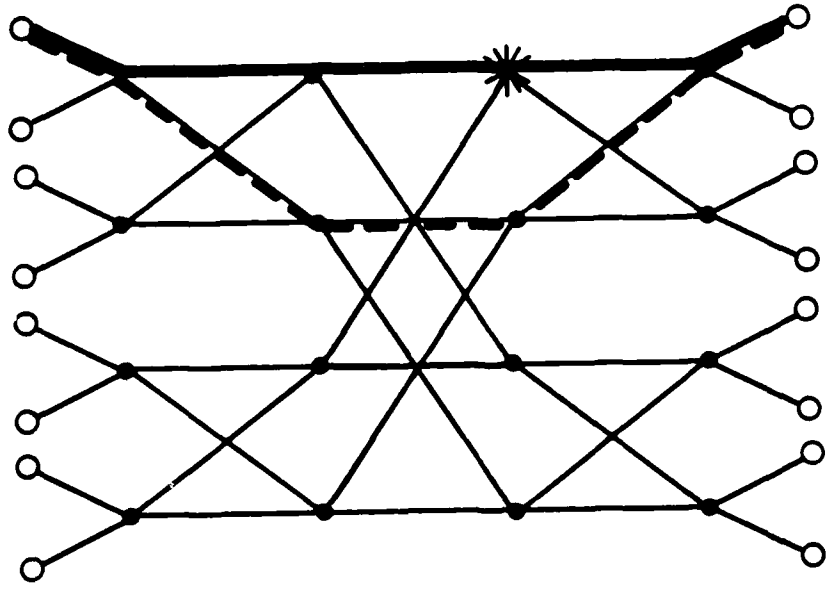
4. Switch Contention

Since there is not an independent path between every possible source and every possible destination in the Butterfly Switch, there is contention for resources. The effect of contention on the behavior of the switch is analysed in detail in Testbed QTR4. This section outlines the mechanisms that the Butterfly Switch uses to deal with contention and summarizes the behavior of the current implementation.

Contention occurs when two messages at a particular Switch Node want to use the same exit path, as shown in Figure 6. In this situation, one message can proceed while the other message cannot. The message that loses can wait until the other has passed, or it can retreat back to its source and retry at a later time.

The retreating strategy was chosen for the current implementation of the Butterfly Switch for two reasons. First, in a unidirectional switch the waiting strategy is not free of deadlocks. This will be discussed further in a later section. Second, a blocked message spends less time in the switch if it retreats. This makes it less likely that the blocked message will obstruct messages that would otherwise have a clear path through the switch. The effective capacity of the switch should therefore be greater when the retreating strategy is used.

When the retreating strategy is used, a message that encounters a conflict must be able to retreat back to its source. This is possible as long as the tail of the message has not left the switch entry port. As a result, the minimum message length for a switch that uses this strategy corresponds to the time that it takes for the head of a message to traverse the switch, plus the interval during which the remote switch interface could reject the message, plus the time that it takes the "retreat" signal to return to the source.



Switch Contention
Figure 6

The choice of contention strategy also impacts the upper limits on the time that it takes for a message to traverse the switch. This worst case time is important because it determines the length of the timeout intervals that are used to detect lost messages. With the wait strategy, the worst case occurs when all sources send messages to the same destination at the same time. The resulting transit time is easy to calculate, and represents a hard upper bound. With the retreating strategy, there is no upper limit. It is possible, though unlikely, to have a transaction that takes five minutes or even a week to reach its destination. However, as long as one is careful to avoid dynamic deadlock situations, it is possible to treat transit times beyond a certain limit as exceptional conditions to be handled in the same manner as message checksum errors and other unlikely but occasional sources of error.

In practice, the round trip time for a 16-bit read operation is 4 microseconds. Given an infinite offered traffic load with a random distribution of destinations, the average reound trip time for the same operation is approximately 12 seconds. This result is derived in QTR4.

5. Clocking and Synchronization

In every digital system, clocking and synchronization issues pervade the designs of the hardware and software. Like most systems, the Butterfly Parallel Processor uses a combination of synchronous and asynchronous techniques. The design is almost entirely synchronous at the lowest level, and every hardware element receives a clock signal from the same central source. The only exception at this level is the system reset signal, which is generated by a toggle switch and synchronized to the system clock at every processor node. At the next level, the system is completely asynchronous, as processors execute independent instruction streams and may send messages through the switch at any time.

6. Important Design Variations

The preceding sections presented the basic structure and operation of the Butterfly Switch. There are also variations that offer a number of implementation choices and connsiderable flexibility in the use of a given switch implementation. This section presents some of the more important variations.

6.1. Parallel Control and Data Paths

The width of the control and data paths in the Butterfly Switch represents a direct tradeoff between three important system parameters: the bandwidth of the switch, the size of the cables that interconnect the Switch Nodes, and the speed and complexity of the logic at each Switch Node. At one extreme, it is possible to have wide data paths and many control lines, yielding high bandwidth with relatively slow and simple logic. At the other extreme, it is possible to use a single wire to carry all of the necessary data and control information. This requires more complex logic to decode the control information, and faster logic to achieve the same bandwidth.

The choice is highly dependent on the implementation technology. The execution speed of the processors and the bandwidth of the memories determine the amount of switch traffic, and hence the switch capacity that is required. The density of the applied technology determines the physical size of the processor and memory modules. This, in turn, determines the amount of cabling that can be sensibly incorporated, and the degree of logic complexity that one can afford at each Switch Node. The speed at which one can drive a single wire determines the number of wires that need to be bundled together to achieve the desired switch capacity.

In the Pluribus multiprocessor, each module occupied a small cage, so it was reasonable to interconnect them with fairly wide paths. In the Butterfly Parallel Processor, the processors are faster and much smaller, so narrower and faster data paths make sense. In the VLSI implemenation of the Butterfly Switch Node, the number of data and control paths is even more important, since it determines the number of pins on the IC package. On the other hand, there is more than enough room on the chip to increase the complexity of the logic. Had backward compatibility not been a requirement, we would have used a smaller number of control and data signals in the VLSI implementation of the Butterfly Switch Node.

6.2. Switch Node Base

So far, all descriptions have assumed that the Switch Nodes have two inputs and two outputs. In fact, it is possible of assign any number of inputs and outputs to a Switch Node. This is the analogous to the notion of the "base" of a Fast Fourier Transform (FFT) implementation, where the base refers to the number of coefficients that are combined in a single Butterfly operation. Here, the term "base" refers to the number of inputs to a given Switch Node.

Unfortunately, the easily recognizable structure of the FFT is lost for bases higher than two. For example, Figure 7 shows the pattern of interconnection for a 16-by-16 Butterly Switch of base 4. Figure 9 shows a 16-by-16 Butterfly Switch of base 2 for comparison. The only change in switch operation brought about by a change in base is the number of header bits used for the routing decision at each Switch Node. A base 4 Switch Node needs two bits to make its routing decision, a base 8 switch needs three bits, and so on. This means that there will be some wasted address bits if the base of a switch is not a power of two.

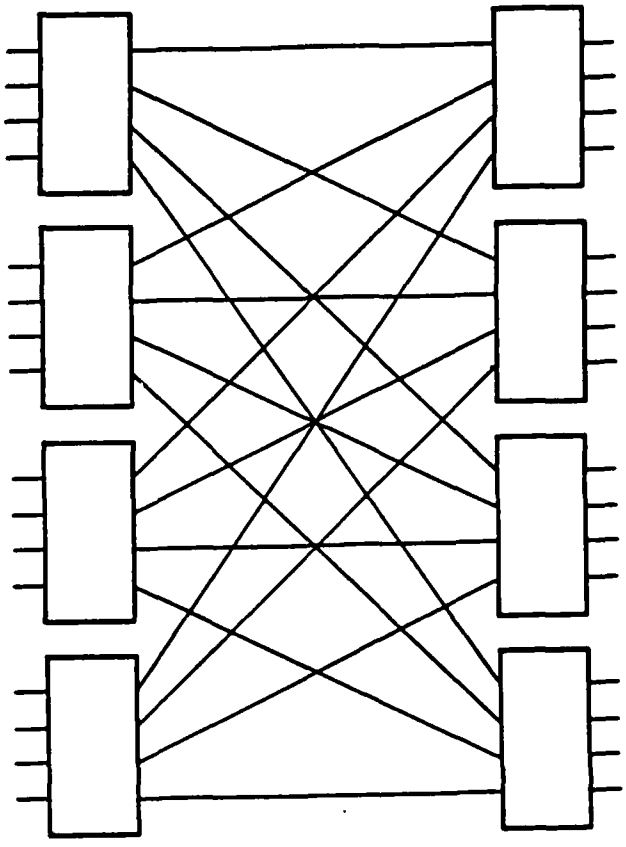
It is clear from the two figures that the switch with the higher base has fewer nodes and fewer wires. This observation can be quantified by giving more exact versions of the formulas alluded to earlier. Given a switch of base B with N input ports and N output ports, the number of columns, wires, and Switch Nodes can be calculated as follows:

$$\text{Columns} = \text{Log}_B(N)$$

$$\text{Wires} = N+N * \text{Log}_B(N)$$

$$\text{Switch Nodes} = (N/B) * \text{Log}_B(N)$$

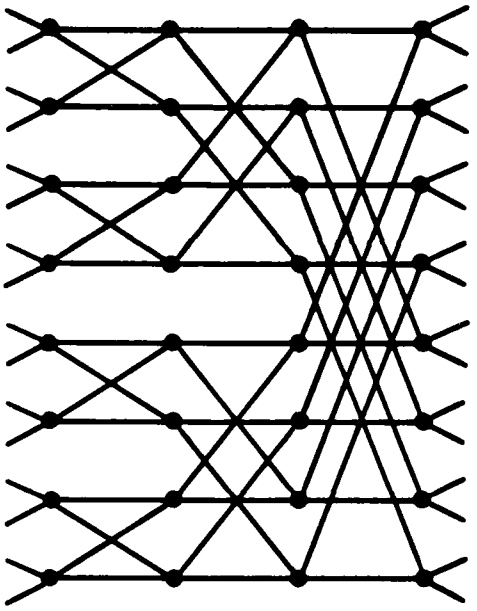
To illustrate the impact of switch base on the implementation cost, Table 1 compares the number of wires and Switch Nodes required to support 256 Processor Nodes for various switch bases.



Base 4 Butterfly Switch
Figure 7

Switch Base	Switch Columns	Nodes	Wires
2	8	1024	2304
4	4	256	1280
16	2	32	768

Comparison of Various Switch Bases
Table 1



Base 2 Butterfly Switch
Figure 8

Switches with large bases have a further advantage because they require fewer columns to implement a switch of a given size. Since collisions in the switch are independent events, the probability that a message will conflict with another message as it traverses the switch is a function of the number of Switch Nodes it must pass through. Thus, for two Butterfly Switch implementations with the same number of ports but different bases, the switch with the larger base will have a lower conflict rate.

Fewer switch columns also means that it takes less time for the head of a message to reach its destination. This parameter is often referred to as the "distance" between an input port and an output port on the switch. Since the data paths through the Butterfly Switch are pipelined, this measure of distance is of minor importance. The transit time of a message in the absence of conflicts

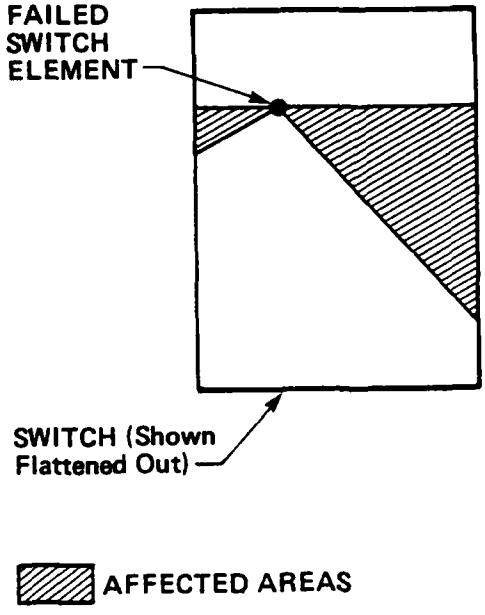
is always dominated by the length of the message. Nevertheless, there is some advantage to having a smaller distance between Switch Nodes, and it is possible to make that distance arbitrarily small by increasing the base of the switch.

All of these considerations suggest that the largest possible base should be selected. However, if we followed this logic to its extreme, we would make every Butterfly Switch a one column switch, which is nothing more than a crossbar. What we have ignored is that each Switch Node is itself a fully connected switch. Thus, the complexity of the Switch Node increases as the square of the switch base.

The Butterfly Switch therefore represents a continuum of implementation choices ranging from the "pure" base 2 Butterfly Switch to a complete crossbar. In practice this means that the choice of switch base is really a packaging decision, influenced by the density of the available technology and the range of machine sizes that one would like to support. In the MSI implementation of the Butterfly Switch, we chose a base 4 switch. This yielded an MSI Switch Node implemenation that fit easily on a small printed circuit board, and provided good conflict statistics over a range of one to 256 Processor Nodes. Backward compatibility dictated the choice of base 4 for the VLSI Switch Node implementation.

6.3. Alternate Paths

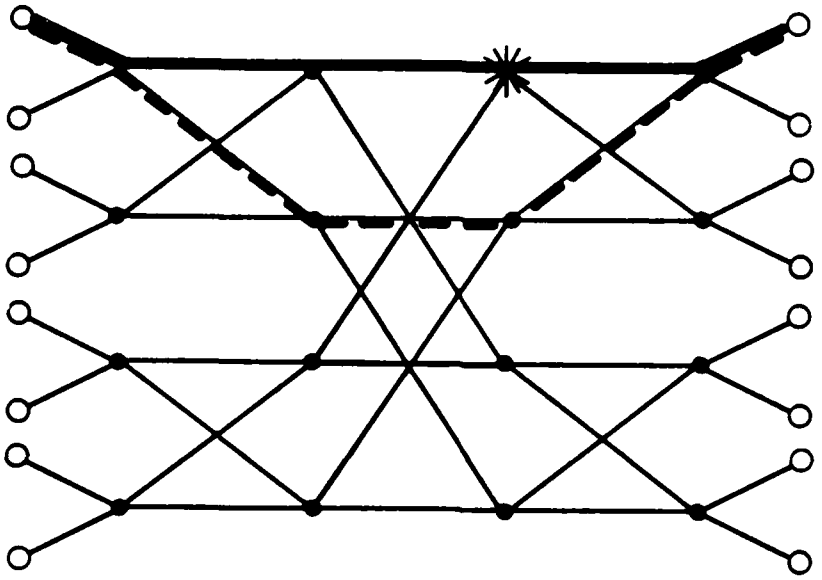
So far, we have described a switch network that is vulnerable to Switch Node failures. Intuitively, the failure of an individual Switch Node can be seen to effect a wedge of Switch Nodes and ports, emanating in both directions from the failed node. This is shown in Figure 9. Outside of the sources and destinations connected via the ports in either the left-hand or the right-hand wedge,



Effect of Switch Failure
Figure 9

the system is fully connected. Thus, the severity of the failure is greater if the failed Switch Node is close to the center of the switch. A failure at the periphery will only effect a few Processor Nodes, while a failure in the interior of the switch is a serious occurrence.

Figure 10 illustrates a simple method for making the switch network immune to single point failures in the interior of the switch. In the figure, an extra column has been added to an eight-port base 2 switch. The resulting switch has two independent paths between every input port and every output port. In other words, this switch configuration gives every destination two addresses, either of which can be used to reach that destination. The high order bit of the message header distinguishes between the two paths. In this configuration, the failure of any node in the interior of



Switch with Alternate Paths
Figure 10

the switch has no effect on the connectivity of the switch. This configuration is still vulnerable to failures in the leftmost and rightmost columns, but the impact of these failures is minor. This increase in reliability is achieved at the relatively low cost of one additional column.

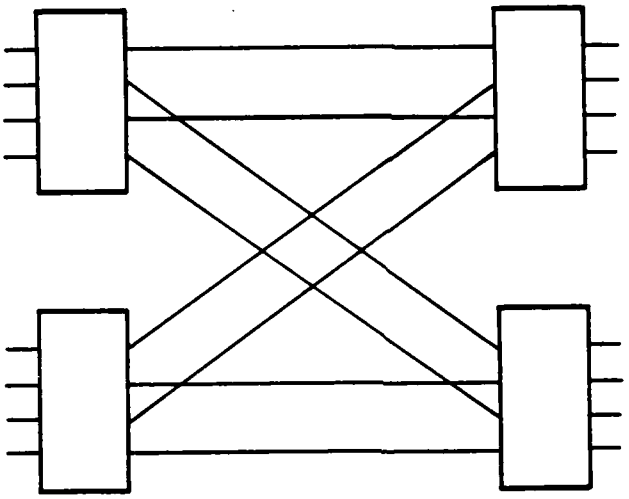
The algorithm for adding the extra column is simple: add the column to the right-hand side of the switch and give it the same connectivity as the leftmost column in the switch. Since the last column is wired in the same manner as the first column, the 3 rightmost columns in Figure 10 implement a complete Butterfly Switch. The decision that is made in the first column of the switch therefore selects one of two entry points into a complete switch, and has no effect on the final destination of the message.

For switches of larger base, the algorithm for adding the column is the same, and the same degree of protection against single point failures is provided. For a switch of base B, the addition of an extra column adds B alternate paths. It is also possible to add more than one extra column. This increases the number of alternate paths and the number of single point failures that the switch can survive.

Since the high order bit (or bits) of a message address distinguishes between alternate paths, the selection of an alternate path is also simple. In the Butterfly Processor Node, this selection is done by the finite state machine that operates the switch interface. When a switch with alternate paths is being used, the finite state machine maintains a two-bit counter that specifies the high order address bits of every message. When a message fails to reach its destination for any reason, the counter is incremented and the message is retransmitted. Thus, there is no overhead assciated with re-routing messages over alternate paths.

6.4. Odd Size Switches

Up to this point, all of the examples shown have used switch sizes that were an even power of the switch base. It is also possible to build "odd size" switches where the number of switch ports is not an even power of the switch base. For larger switches, there are several interesting odd sizes. For example, the eight-port base 4 switch shown in Figure 11 uses four Switch Nodes (exactly half the number in a 16-port switch) and is wired to provide two alternate paths between every source and every destination. Similarly, 64- and 128-processor machines can be wired to have alternate paths.



Eight Port Switch
Figure 11

6.5. Message Size

Since the Butterfly Switch is pipelined, it is reasonable to expect that messages experience some initial setup delay, followed by a high transfer rate. This means that the Butterfly Switch favors long messages over short messages. While this is true of most communications systems, regardless of their architecture, a tightly coupled parallel processor must read and write single words as well as large blocks of data. Butterfly Processor Nodes are therefore capable of executing both single word and multiple word transactions.

The presence of long messages in the switch can adversely effect the latency of the short messages. A transfer of 64 kilobytes (the maximum length allowed in the current system) takes about 16 milliseconds to complete. During the transfer, one path through the switch is constantly

occupied. To avoid this problem, multiple word transfers are broken into shorter blocks by the Processor Node software. Although we have not studied this issue in detail, the system performs quite well with a block length of 256 bytes.

6.6. Bidirectional Switch

In the current implementation of the Butterfly Switch, the request and reply messages for two-way transactions, such as memory read operations, are completely independent. The switch could also have been constructed so that the path established by the request message is held open for the reply to return over the same path. With the latter scheme, the switch paths would be used first to establish a path and to send the request message. After the request has been processed, the direction of data flow would be reversed and the reply message sent back over the same path. We refer to this design as the "bidirectional switch." There are several advantages to this kind of switch:

1. While the request message suffers from conflicts in the switch, the reply message does not.
2. It is not necessary to include the address of the sending Processor Node in the request message, since a path for the reply has already been established.
3. The overhead associated with setting up and initiating a message can be much smaller for the reply.
4. The bidirectional switch is free of deadlocks when the wait strategy is used, and less prone to dynamic deadlocks when the retreating strategy is used.

The bidirectional switch also has some potential drawbacks:

1. The average profile of a transaction is potentially increased because paths through the switch are held at times when no data is being transferred.

2. The complexity of the Switch Node must be increased to handle bidirectional traffic.

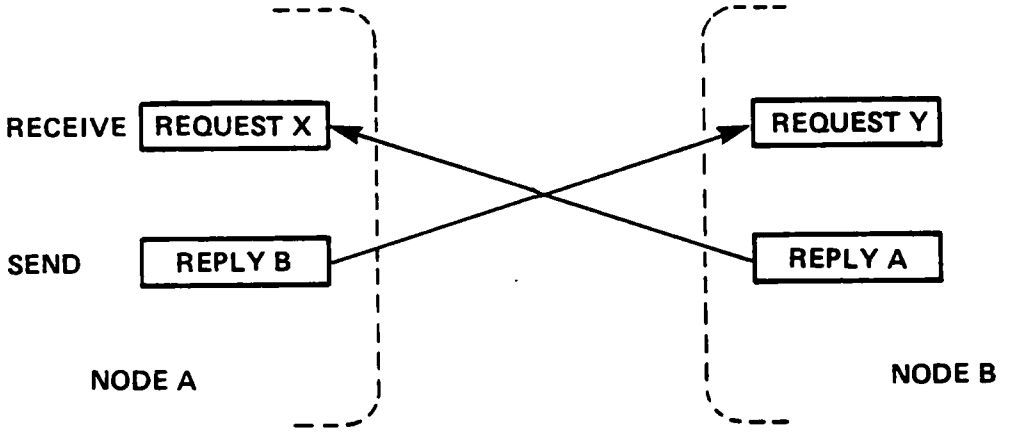
3. The path must be held while the destination is generating its reply.

While the possibilities offered by the bidirectional switch are intriguing, we have not implemented one, so we cannot say much more about its properties at this point.

7. Deadlocks

In any complex computer system, there are opportunities for deadlock. The Butterfly Parallel Processor is no exception. One fairly simple example is illustrated in Figure 12, which shows two Processor Nodes, each with a single input buffer and a single output buffer. In this example, Processor Node A has sent a request message Processor Node B. After the request was received, but before the reply could be sent, a request from Processor Node C arrives at Processor Node A. At this point, Node A cannot receive a reply from Node B because its input buffer is full, and Node A cannot service the request from Node C until it receives a reply from Node B.

To avoid this kind of deadlock, the system must ensure that reply messages can always be transmitted and accepted. To do this, we use a simple buffer allocation strategy that is reminiscent of the buffer management strategies used in the ARPANET IMP [ROS80] and other packet switching systems.



Switch Deadlock
Figure 12

On the receive side, each Processor Node has two buffers: one for requests and one for replies. In addition, the switch receiver is allowed to examine the first few bits of a message to determine whether it is a request or a reply. If the incoming message is a request, and the request buffer is full, the incoming message is rejected and must be retransmitted. Note that this scheme does not prevent deadlocks in a switch that uses the wait strategy. Once a request arrives at an input port it will block the path to the input port until it is accepted by the destination Processor Node.

A similar strategy can be applied on the transmit side to ensure that resources are always available for the transmission of reply messages. Each Processor Node has two transmit buffers: one for requests and one for replies. Whenever a request message is rejected, the transmitter makes sure that the reply buffer is empty before it tries to send the request again.

The bidirectional switch is free from all of these deadlocks because it pre-allocates a path through the switch for the reply. Furthermore, the transmit side is independent of the corresponding request buffer, when a bidirectional switch is used.

8. Flow Control

The memory system on each Processor Node services requests from the local 68000, the local I/O bus, and the switch interface. The bandwidth of the memory system is 40 megabits per second, allowing it to keep up with any one of these elements, but not all three at the same time. Each element is therefore given a fixed priority by the memory system, with the I/O bus at the highest priority level and the 68000 at the lowest level.

Since most transactions between Processor Nodes involve the exchange of short request and reply messages, memory access delays due to I/O activity usually mean that a request or reply message will be held in the appropriate switch interface buffer for a slightly longer period of time. For block transfer operations, however, it is necessary to invoke a flow control mechanism when the memory is occupied by I/O traffic. Thus, the switch interface on every Processor Node is capable of slowing down the rate at which it delivers or accepts switch data. The details of this mechanism are explained in a later section.

9. Error Control

As in any communications network, message packets passing through the Butterfly Switch may arrive at their destinations with bit errors. These errors may be due to hardware failure or to

transient conditions such as power line glitches.

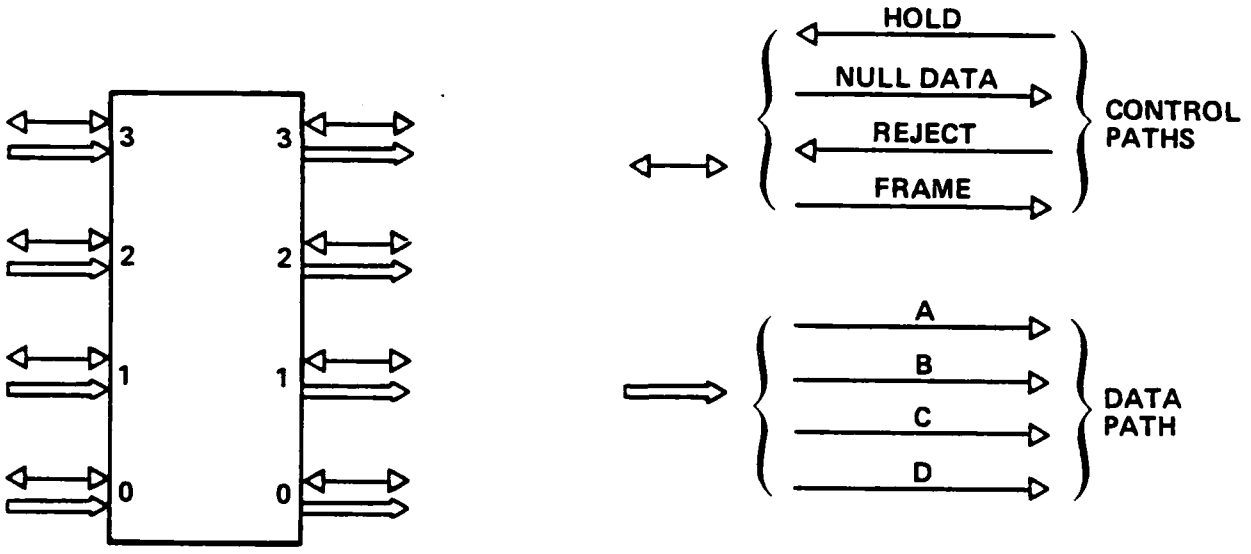
To minimize the occurrence of errors due to transient conditions, the communications hardware in the switch is designed to be very robust. All signals that run between boards use balanced ECL line drivers and receivers, the system uses a fully synchronous clocking methodology with conservative timing margins, and every board has its own switching regulator to provide well-controlled, noise-free power.

To detect systematic errors due to hardware failures, every packet that passes through the switch carries a checksum. The checksum will not detect every bit error, but it reliably detects the gross errors that occur as a result of a loose connector or a failed driver. When a message arrives with a bad checksum, the message is ignored by the receiving Processor Node and the occurrence of the error is flagged in a register that is monitored by the Operating System.

10. Switch Node Design

This section decribes the current hardware implementation of the Butterfly Switch. As described earlier, the switch comprises a collection of interconnected Switch Nodes. In the initial implementation, each Switch Node was a self-contained 12" by 12" printed circuit board with a switching power supply, an MSI implementation of the routing algorithm, and a balanced ECL interface to neighboring nodes. In the current implementation, the MSI implemenation of the routing algorithm has been replaced by a custom NMOS chip. This allows us to put put eight Switch Nodes on a single printed circuit board.

In effect, each Switch Node is an "intelligent" 4-by-4 crossbar. For each packet that enters the node, the first two bits are used to select an output port, and the remainder of the packet is routed through that port. If the output port is busy when the packet arrives, a "reject" signal is propagated back to the source of the packet. In addition to its routing functions, the Switch Node implements a round-robin priority mechanism to deal with contention for output ports. It also supports end-to-end flow control through the switch.



Switch Node Block Diagram
Figure 13

As shown in Figure 14, each channel through the VLSI Switch Node is composed of eight signals. Four data lines (D0-D3) along with two control signals (FRAME and NULL_DATA) carry information towards the message destination. Two additional control signals (REJECT and HOLD) carry information from the destination to the source.

The data signals carry the routing information at the beginning of the packet, followed by the data that forms the body of the packet, and then a four-bit checksum. Since the data path is four bits wide, one nibble of data is transmitted on every clock tick.

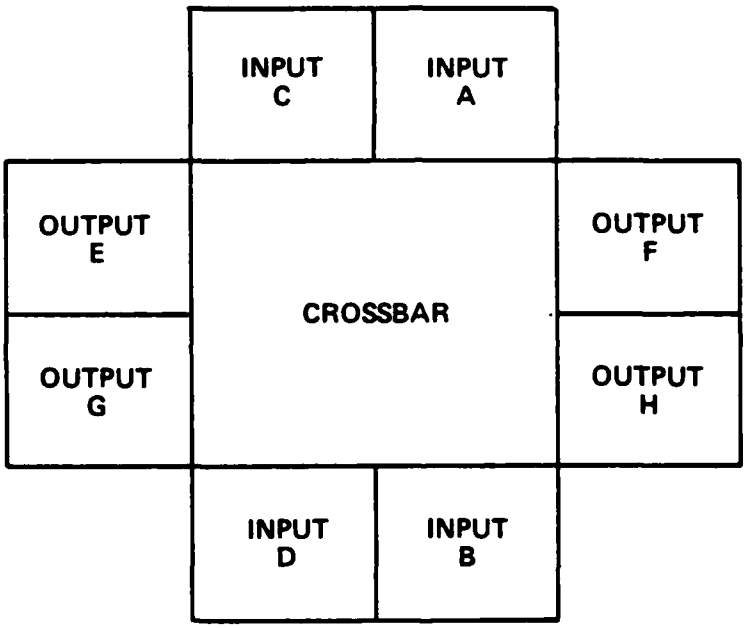
The FRAME signal accompanies each packet and serves to define its head and tail. As the head of a message enters a Switch Node, a crosspoint link is formed so that the message can pass through. As the tail leaves the node, the link is broken.

The REJECT signal is generated if a Switch Node cannot establish the required crossbar link when the message head appears. The REJECT signal travels back towards the source of the message, breaking the crossbar links at each Switch Node as it goes and directing the source Processor Node to try again at a later time. A REJECT signal can also be generated by the destination Processor Node if it decides not to accept the message.

The HOLD and NULL data signals are used for flow control during the transmission of long switch messages. When the switch input FIFO on a destination Processor Node fills beyond a certain point, it asserts the HOLD signal. The HOLD signal propagates back to the Source Processor Node at a rate of one Switch Node per clock tick.

Since the HOLD signal may be asserted quite frequently, it would be inefficient for the source Processor Node to release the switch every time the destination processor gets behind. Instead, it keeps the connection open, transmits a nonsense data pattern, and asserts the NULL_DATA signal to

indicate to the Destination Processor that the incoming data can be ignored. When the HOLD signal is deasserted, the Source Processor Node deasserts the NULL_DATA signal and starts transmitting message data again. The detailed operation of the NULL_DATA signal is actually somewhat more complicated than this. However, none of these details effect the operation of the switch, so they are not described here.



Butterfly Switch Chip Floor Plan
Figure 14

As shown in Figure 15, the physical organization of the switch chip reflects its logical organization. The chip consists of eight finite state machines (FSMs) clustered around a crossbar network. Each input and output port is associated with a different FSM. When a packet arrives, an input FSM uses the low order bits of the first nibble of data to decide which output port to use. On

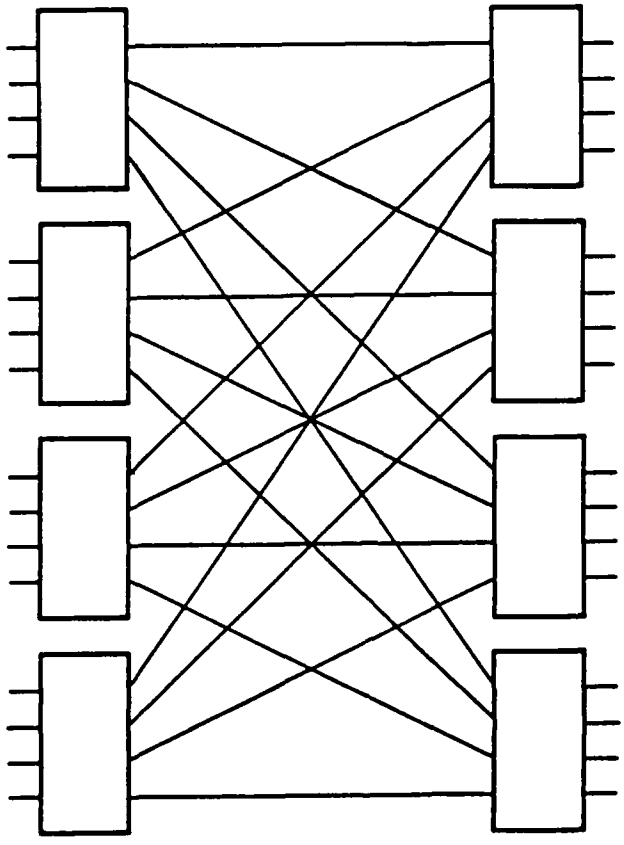
the next cycle, the input FSM "bids" for the output port by driving a line connected to the selected output FSM. If the output port is not busy and if no other message having higher priority is simultaneously bidding for the same port, the output FSM opens a set of pass transistors at the appropriate set of crosspoints in the crossbar. This forms a link between the input port and the output port. Otherwise, the output FSM asserts a REJECT signal that propagates back through the input FSM to the originator of the message. Since the first nibble of data is not passed on to the next Switch Node, a full clock cycle is available to make the routing decision and to establish the crossbar link.

When two input ports make simultaneous bids for the same output port, the output port must decide which one to reject. Since messages arrive at random times and retreat when they are rejected, the method used to make this decision is not very important to the performance or fairness characteristics of the Butterfly Switch as a whole. The method used in the Butterfly Switch chip is based on a simple round-robin priority scheme. In this scheme, the priority ordering is determined by the input port that was granted access most recently. As shown in Table 2, the most recently granted port has lowest priority, and the ordering of the remaining ports follows a fixed pattern.

Most Recent User:	A	B	C	D
Highest:	B	C	D	A
	C	D	A	B
	D	A	B	C
Lowest:	A	B	C	D

Output Port Priority Assignments
Table 2

The Butterfly VLSI Switch Node (BVSN) uses eight switch chips to implement a 16-port Butterfly Switch on a single printed circuit board. A block diagram is shown in Figure 15. The BVSN can be used by itself to support a 16-processor Butterfly, or it can be wired together with other BVSNs to form a larger switch. Like all Butterfly boards, it accepts unregulated power at 30 volts and uses an onboard switching supply to generate local voltages. The board is 12" by 18" (the same size as a Butterfly Processor Node). A 3" by 19" metal panel mounted at the front of the board provides mechanical support for 32 I/O connectors. The equivalent MSI implementation uses 1152 square inches of PC board area (versus 216) and 192 square inches of rack space (versus 63). It is interesting to note that only 15% of the BVSN is occupied by switch chips. The remainder of the board is occupied by drivers and receivers (40%), power supply (25%), connectors (15%), and local interconnect between switch chips (5%). On the average, the cost of the switch (including BVSN boards and cables) accounts for approximately 15% of the hardware cost of a Butterfly system.



BVSN Block Diagram
Figure 15

REFERENCES

[ABRA69] N. Abramson, "The ALOHA System – Another Alternative for Computer Communications" AFIPS Conference Proceedings 37, 1969.

[HEAR73] F.E. Heart, S.M. Ornstein, W.R. Crowther, W.B. Barker, "A New Minicomputer/Multiprocessor for the ARPA Network" AFIPS Conference Proceedings 42, June 1973.

[IEEE83] "IEEE Standard Microcomputer System Bus" IEEE std 796-1983 Institute of Electrical and Electronics Engineers Inc. 1983.

[ROS80] E. Rosen, "Issues in Buffer Management" Internet Experiment Note #182, 1980.

END

FILMED

5-86

DTIC